**Exp 2:**

To issue a command, start a "Terminal" (for Ubuntu/Mac) or "Git Bash" (for Windows):

$ **git <command> <arguments>**

The commonly-used commands are:

1. **init**, **clone**, **config**: for starting a Git-managed project.
2. **add**, **mv**, **rm**: for staging file changes.
3. **commit**, **rebase**, **reset**, **tag**:
4. **status**, **log**, **diff**, **grep**, **show**: show status
5. **checkout**, **branch**, **merge**, **push**, **fetch**, **pull**

**Step1:** Download & Install:

* For Ubuntu, issue command " sudo apt-get install git".

To check version of git:

#git –version

**Step 2**: Customize and configure your Git Account:

#git config --global user.name “username”  
#git config --global user.email “email-id”

To List Global configuration for Git:

#git config –list

step 3 : create a repo in github and clone it using following command

#git init  
#git clone “url.git”  
#ls  
create a file and write a code in it  
#git status  
# git add filename  
for commit  
#git commit -m “msg”

Push cmd  
# git push origin main

To see the logs use cmd  
# git log  
# git show [for showing repo id and other details]

# git diff [to see the diff in content of file bet 1 and 2 commit]

To revert intial commit use cmd  
# git revert <initial commit id>

**Exp3:**

Aim: Understand and perform version control and source code management using Git.

**Branching and Merging**

**Steps:**

1. **Create a New Branch:**  
   Use git branch <branch-name> to create a branch.
2. **Verify the Current Branch:**  
   Use git branch to view all branches and see the current one.
3. **Checkout the New Branch:**  
   Use git checkout <branch-name> to switch to the new branch.
4. **Stage Changes:**  
   After editing files, use git add <file> and verify using git status.
5. **Commit the Changes:**  
   Use git commit to commit the staged files. View commit history using git log.
6. **Merge the Changes:**  
   Switch back to the main branch (git checkout main) and merge the changes from the feature branch (git merge <branch-name>).
7. **Push the Changes to Remote Repo:**  
   Push the changes from your local repository to a remote one.
8. **Delete the Branch:**  
   After merging, delete the branch using git branch -d <branch-name>

 Delete the branch using the # **git branch -d** <branch-name> command

**Exp 4:**

Aim:

Install and configure Jenkins to test and deploy an application using Maven.

Steps:

1. Update and Install Required Software

* sudo apt-get update: Update Ubuntu repository.
* sudo apt-get install openjdk-11-jdk: Install Java Development Kit (JDK).
* java -version: Verify Java installation.

2. Install Jenkins

* Install Jenkins using these commands:

bash

Copy code

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins

* jenkins --version: Check if Jenkins is installed.

3. Start Jenkins

* sudo systemctl start jenkins.service: Start Jenkins service.
* sudo systemctl status jenkins: Check Jenkins status (should be active/running).

4. Configure Firewall

* Allow Jenkins to run on port 8080 by adjusting the firewall using:

bash

Copy code

sudo ufw allow 8080

5. Access Jenkins Setup

* Open http://localhost:8080 or http://127.0.0.1:8080 in a browser to start Jenkins setup.
* Use cat /var/lib/jenkins/secrets/initialAdminPassword to retrieve the initial admin password.

6. Customize Jenkins

* Select "Install suggested plugins" during setup.
* Create an admin user by specifying username, password, and email.

Creating and Running a Jenkins Job:

1. Create a New Job

* From the Jenkins dashboard, click New Item.
* Enter an item name (e.g., *HelloWorld*), select Freestyle project, and click OK.

2. Configure Git for the Project

* Under Source Code Management, select Git.
* Set up a GitHub repository for your Java project:
  + git init: Initialize the Git repository.
  + git add .: Stage all files.
  + git commit -m "Initial commit": Commit the changes.
  + git remote add origin <repo-url>: Add the remote repository URL.
  + git push -u origin master: Push the files to GitHub.

3. Build the Project

* Under Build, add a build step: Execute Shell.
* Enter the following commands to compile and run Java code:

bash

Copy code

javac HelloWorld.java

java HelloWorld

4. Execute the Build

* Click Build Now to run the build.
* Check the build status in Build History (blue = success, red = failed).
* View build details and console output.

**Exp 5:**

**Aim:**

Create and build a CI/CD pipeline in Jenkins to test and deploy an application over a Tomcat server.

**Steps:**

**1. Configuring Jenkins**

* Find Java location:

bash

Copy code

find / -type f -name java

* Set Java and Maven paths in Jenkins under **Global Tool Configuration**.

**2. Create a CI/CD Pipeline in Jenkins**

* **Step 1:** Create a new item (Freestyle project) in Jenkins.
* **Step 2:** In the Source Code Management section, paste the GitHub repository link and specify the branch name.
* **Step 3:** Click **Build Now** to fetch repository files.
* **Step 4:** In the **Build** section, add a build step and choose **Invoke top-level Maven targets**.
* **Step 5:** Select Maven goals (e.g., clean install) and save. Build again to generate a WAR file.

**3. Deploying to Tomcat**

* **Step 6:** Check for the WAR file in the Jenkins workspace or server directory.
* **Step 7:** In the **Post-build Actions** tab, select **Archive the Artifacts** and specify \*\*/\*.war.
* **Step 8:** Install Apache Tomcat on your server:

bash

Copy code

sudo groupadd tomcat

sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat

curl -O https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.93/bin/apache-tomcat-9.0.93.tar.gz

sudo mkdir /opt/tomcat

sudo tar xzvf apache-tomcat-9.0.93.tar.gz -C /opt/tomcat --strip-components=1

* **Step 9:** Update Tomcat’s server.xml to change the port from 8080 to 8090.
* **Step 10:** Update tomcat-users.xml to configure roles and users for deployment.
* **Step 11:** Remove IP restrictions in context.xml to allow access to the Tomcat Manager.

**4. Deploy via Jenkins**

* **Step 12:** Restart Tomcat:

bash

Copy code

./shutdown.sh

./startup.sh

* **Step 13:** In Jenkins, install the **Deploy to Container** plugin.
* **Step 14:** Add the **Deploy WAR/EAR to a container** post-build action, select **Tomcat 9.x Remote**, and provide credentials.

**5. Final Deployment**

* **Step 15:** Click **Build Now** to deploy the WAR file to the Tomcat server.

**Exp 6 :**

**Aim:**

To implement a Jenkins Master-Slave Architecture with scaling.

**Steps to Configure Jenkins Master and Slave Nodes:**

1. **Access Jenkins Dashboard**: Go to **Manage Jenkins** -> **Manage Nodes**.
2. **Create New Node**: Enter the node name, choose **Permanent Agent**, and configure it.
3. **Configure Node Details**: Set up JDK path under 'Node Properties.'
4. **Configure JNLP Port**: Open port 50000 on the firewall to allow connections between master and slave nodes.
5. **Establish Connection**: Run commands on the slave node to connect it with the master using JNLP.
6. **Job Configuration**: Mention the slave node’s label in the job configuration on the Jenkins dashboard.
7. **Run the Job**: Configure and execute jobs using the slave node.

Exp 7:

**Aim:**

To implement Selenium automation.

**Theory:**

* **Selenium**: An open-source Web UI automation testing tool, supporting multiple browsers, platforms, and programming languages (e.g., Java, C#, Python).
* **Compatibility**: Can be deployed on various operating systems (Windows, Linux, Mac, iOS, Android).
* **Integration**: Can integrate with tools like Maven, Jenkins, Docker for continuous testing, and TestNG/JUnit for test management and report generation.
* **Automation Testing**: Automates execution of test cases without human intervention, generating detailed reports.

**Steps for Selenium Automation in DevOps on Ubuntu:**

1. **Install Selenium IDE**:
   * Available as Firefox and Chrome plugins.
   * Install it by adding the extension and restart the browser to activate.
2. **Create a Basic Test Case in Selenium IDE**:
   * **Record**: Start recording user interactions like navigating to a URL, searching, and clicking links.
   * **Playback**: Execute the recorded script using the "Run Current Test" button to verify actions.
   * **Save**: Save the test suite in .side format for future use

Exp 8:

**Aim:**

To demonstrate the container lifecycle using various Docker commands.

**Steps for Docker Container Lifecycle:**

1. **Install Docker**:
   * Run the Docker installation script:

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

* + Verify installation with:

#docker version

1. **Basic Docker Commands**:
   * **Login**:

docker login

* + **Run a Container**:

docker run hello-world

docker run -it ubuntu bash

* + **View Docker Images**:

docker images

* + **Pull an Image**:

docker pull mysql

* + **View Running Containers**:

docker ps

* + **View All Containers (Running or Exited)**:

docker ps -a

1. **Container Management Commands**:
   * **Execute a Container**:

docker exec -it <container\_id> bash

* + **Delete a Container**:

docker rm <container\_name/container\_id>

* + **Delete an Image**:

docker rmi <image\_name/image\_id>

1. **Container Lifecycle Commands**:
   * **Start, Stop, Pause, and Kill Containers**:

docker start <container\_id>

docker stop <container\_id>

docker pause <container\_id>

docker kill <container\_id>

**Setting Up a MySQL Database Inside Docker:**

1. **Pull MySQL Image**:

docker pull mysql:latest

1. **Run MySQL Container**:

docker run --name test-mysql -e MYSQL\_ROOT\_PASSWORD=strong\_password -d mysql

1. **Access MySQL Terminal**:

docker exec -it test-mysql bash

mysql -u root -p

Exp 9:

**Aim:**

Build a Docker image for a sample web application using various Dockerfile instructions.

**Theory:**

A **Dockerfile** contains commands to create a Docker image, which is a portable and lightweight executable package including all software dependencies. A **container** is a runtime instance of an image, making software deployment efficient and isolated.

**Dockerfile Commands:**

1. **FROM**: Specifies the base image (OS).
   * FROM <ImageName>
2. **COPY**: Copies files/folders into the image during build.
   * COPY <Source> <Destination>
3. **RUN**: Executes scripts/commands while creating the image.
   * RUN <Command + ARGS>
4. **CMD**: Runs a command inside the container.
   * CMD [command + args]

**Steps to Create a Docker Image:**

1. Create a Dockerfile.
2. Add necessary instructions.
3. Build the Dockerfile to generate an image.
4. Run the image to create a container.

**Part I: Containerizing with Docker CLI**

1. Pull the **nginx** image: docker pull nginx
2. Run the container: docker run --name <container\_name> <image\_id>
3. Modify and deploy a static website within the container.

**Part II: Creating a Dockerfile**

1. Write a **Dockerfile** for the application.
2. Build the image: docker build -t <dockerhub\_username>/<image\_name> .
3. Run the container: docker run -p <port>:80 <image\_name>
4. Optionally, push the image to Docker Hub for sharing.

Exp 10:

**Aim:**

Install Ansible on an AWS instance, configure SSH access between the Ansible master and slave, and test the connection.

**Theory:**

Ansible is an open-source IT automation tool used for configuration management and orchestration. It includes components such as:

* **Controller Machine**: Where Ansible is installed.
* **Inventory**: A file listing the servers.
* **Playbook**: Defines automation tasks in YAML format.
* **Modules**: System tasks like file handling or package management.
* **Roles**: Organizing playbooks for reuse.
* **Handlers**: Triggering service changes (e.g., restarting services).

**Steps:**

1. **Connect AWS Instances**  
   Ensure that AWS instances for master and slave are connected.
2. **Ansible Installation on Master**

# apt update -y

# apt-add-repository ppa:ansible/ansible

# apt update -y

# apt-get install ansible -y

# ansible --version

1. **Update Slave**

# apt update -y

1. **Configure Hosts File on Master**
   * Edit the Ansible hosts file to include the slave IP.

# nano /etc/ansible/hosts

1. **Create SSH Key for Ansible Master-Slave Communication**
   * On the master:

# ssh-keygen -t rsa

# cat /root/.ssh/id\_rsa.pub

* + Copy the public key to the slave’s authorized\_keys file.

1. **Test Connection**
   * On the master:

# ansible -m ping all

1. **Install and Remove Packages on Slave via Ansible**
   * Install git:

# ansible client\_1 -m apt -a "name=git state=present" --become

* + Uninstall git:

# apt remove git

1. **Editing Files via Ansible**
   * Uninstall nano and install it again via Ansible:

# ansible client\_1 -m apt -a "name=nano state=absent" --become

# ansible client\_1 -m apt -a "name=nano state=present" --become